Meta-Surrogate Benchmarking for Hyperparameter Optimization

Aaron Klein
Department of Computer Science
University of Freiburg
kleinaa@cs.uni-freiburg.de

Zhenwen Dai
Amazon
Cambridge, United Kingdom
zhenwend@amazon.co.uk

Frank Hutter
Department of Computer Science
University of Freiburg
fh@cs.uni-freiburg.de

Neil Lawrence
Amazon
Cambridge, United Kingdom
lawrennd@amazon.co.uk

Javier Gonzalez
Amazon
Cambridge, United Kingdom
gojav@amazon.co.uk

Abstract

Despite the recent progress in hyperparameter optimization (HPO), available benchmarks that resemble real-world scenarios usually consist of a few and very large problem instances that are expensive to solve. This blocks researchers and practitioners from systematically running large-scale comparisons that are needed to draw statistically significant results. This work proposes a method to alleviate these issues by means of a meta-surrogate model for HPO tasks trained on off-line generated data. The model combines a probabilistic encoder with a multi-task Bayesian optimization model such that it can generate inexpensive and realistic tasks of the class of problems of interest. We demonstrate that benchmarking HPO methods on samples of the generative model allows us to draw more coherent and statistically significant conclusions that can be reached orders of magnitude faster than using the original instances. We provide evidence of our findings for a large variety of HPO methods on a wide class of problems.

1 Introduction

Automated Machine Learning (AutoML), (Hutter et al., 2018) is an emerging field that studies the progressive automation of machine learning. It has already shown promising results by outperforming human experts in finding better hyperparameters (Snoek et al., 2012), an thereby for example substantially improved AlphaGo (Chen et al., 2018), or neural network architectures (Zoph and Le, 2017; Real et al., 2017).

A core part of an automated machine learning system is the hyperparameter optimization (HPO) of a machine learning algorithm. Despite recent progress, during the phases of developing and evaluating new HPO methods one frequently faces the following problems:

- Evaluating the objective function used to tune machine learning algorithms is often expensive in terms of wall-clock time; *e.g.*, the evaluation of a single hyperparameter configuration

Preprint. Under review.

may take several hours or days. This makes extensive hyperparameter search or repeated runs of HPO methods computationally infeasible.

- Even though repositories of datasets such as OpenML (Vanschoren et al., 2014) provide thousands of datasets, a large fraction cannot meaningfully be used for HPO since they are too small or too easy (in the sense that even simple methods achieve top performance). Hence, useful available datasets are scarce, making it hard to produce a comprehensive evaluation of how well a HPO method will generalize across tasks.

These two problems make benchmarking of HPO techniques extremely challenging. Due to scarce data and expensive objective functions, there are a limited amount of comparisons that researchers can carry out within a reasonable computational budget. This delays the progress of the field as statistically significant conclusions about the performance of different HPO methods for classes of problems of interests may not be possible to draw. See Figure 2 for an illustrative experiment. It is well known that Bayesian optimization (Shahriari et al., 2016) is better than naive random sampling in terms of the number of evaluations required to identify the best regions to optimize an objective function. While we show clear evidence for this in Appendix B on a larger set of datasets, this conclusion cannot be reached when optimizing hyperparameters of XGBoost (Chen and Guestrin, 2016) on the three "unlucky" picked datasets in Figure 2. Surprisingly, the community has not paid much attention to this issue of proper benchmarking, which is a key step required to generate the scientific knowledge needed to make informed decisions about how machine learning should be deployed in the real world.

In this work we present a generative meta-model that, when conditioned on off-line generated data, allows the sampling of an unlimited number of new tasks that share properties with the original ones. There are several advantages to this approach. First, the generated problem instances are inexpensive to evaluate as they are synthetically generated with a closed-form. This approach drastically reduces the resources needed to compare HPO methods, bounded only by the optimizer's computational overhead. See Figure 1 for an illustration of the differences between benckmarking on real problems vs. synthetically generated samples. Second, there is no limit in the number of tasks that can be generated, which helps to discover the right statistical significance in comparison tests. Third, the shape and properties of the tasks are not predefined but learned using a few real task of an HPO problem. While the *global* properties of the initial tasks are preserved in the samples, the generative model allows the exploration of instances with diverse *local* properties making comparisons more robust and reliable. See Appendix D for some 2 dimensional example tasks.

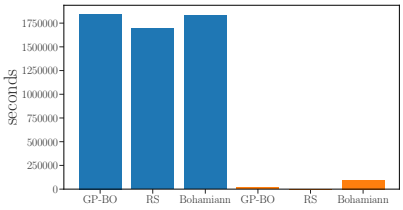

Figure 1: The three blue bars on the left show the total wall-clock time of executing 20 independent runs of GP-BO, RS and Bohamiann with 100 function evaluation for the HPO of a feed forward neural network on MNIST. The orange bars show the same for optimizing a tasks sampled from PROFET. Benchmarking with PROFET is orders of magnitude cheaper in terms of wall-clock time than the original benchmarks, thereby the computational time is almost exclusively spend for the optimizer overhead (hence the larger bars for GP-BO and Bohamiann compared to RS).

2 Related Work

The use of meta-models that learn across tasks has been investigated by others before. With the purpose of warm-starting the HPO on new tasks from previously optimized tasks, Swersky et al. (2013) extended Bayesian optimization to the multi-task setting by using a Gaussian process that also models the correlation between tasks. Instead of a Gaussian process, Springenberg et al. (2016) used a Bayesian neural network inside multi-task Bayesian optimization that automatically learns an embedding of task during optimization. Perrone et al. (2018) used Bayesian linear regression, where the basis functions are learned by a neural network, to warm start Bayesian optimization from previous tasks. Feurer et al. (2015b) used a set of dataset statistics as meta-features to measure the similarity between tasks, such that hyperparameter configurations that were superior on previously optimized similar tasks can be evaluated as part of the initial design before the actual Bayesian optimization procedure starts. This technique is also applied inside the auto-sklearn framework (Feurer et al.,

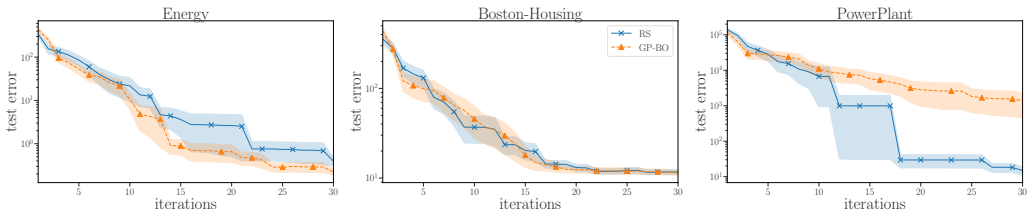

Figure 2: Common pitfalls in the evaluation of HPO methods: we compare two different HPO methods (Bayesian optimization with Gaussian processes and random search) for optimizing the hyperparameters of XGBoost on three UCI regression datasets (see Section B in supplementary materials for more datasets). The small number of tasks makes it hard to draw any conclusions, since the ranking between the methods varies between the tasks. Furthermore, a full run might takes several hours which makes it prohibitively expensive to average across a large number of runs.

2015a). van Rijn and Hutter (2018) evaluated random hyperparameter configurations on many different tasks to learn priors for a support vector machine, random forest and Adaboost. The idea of using a latent variable to represent correlation among multiple outputs of Gaussian process has been exploited by Dai et al. (2017).

In the context of benchmarking HPO methods, HPOlib (Eggensperger et al., 2013) is a benchmarking library that provides a fixed and rather small set of problems that have been used to compare several Bayesian optimization tools. In earlier work, Eggensperger et al. (2015) also used surrogates to speed up the empirical benchmarking of HPO methods. Similar to our work, these surrogates are trained on data generated in an off-line step. Afterwards, function evaluations require only prediction of the surrogate model instead of actually running the benchmark. However, these surrogates only mimic one particular task and do not allow for generating new tasks as presented in this work. Recently, tabular benchmarks were introduced for neural architecture search (Ying et al., 2019) and hyperparameter optimization (**?**) problems, which first perform an exhaustive search of a discrete benchmark problem to store all results in a database and then replace expensive function evaluations by efficient table lookups. While this does not introduce any bias due to the model (see Section 6 for a more detailed discussion), tabular benchmarks are only applicable for discrete optimization problems that consist of a small configuration space.

Related to our work but for benchmarking general blackbox optimization methods is the COCO platform (Hansen et al., 2016b). The difference with our approach is that is based on handcrafted synthetic functions that do not resemble real world HPO problems.

3 Benchmarking HPO methods with generative models

We now describe the generative meta-model to generate HPO tasks. First we give a formal definition about how to benchmark optimization methods across tasks sampled from a unknown distribution and then we describe how we can approximate this distribution by our meta-model, such that we can sample new tasks in a parameteric form.

3.1 Problem Definition

We denote $t_1, \ldots, t_M$ to be a set of related objectives/tasks with the same input domain $\mathcal{X}$. We assume that each t_i for $i = 1, \ldots M$, is an instantiation of an unknown distribution of tasks $t_i \sim p(t)$. Every task t has an associated objective function $f_t : \mathcal{X} \subset \mathbb{R}^d \rightarrow \mathbb{R}$ where $\boldsymbol{x} \in \mathcal{X}$ represents a hyperparameter configuration and we assume that we can observe f_t only through noise: $y_t \sim \mathcal{N}(f_t(\boldsymbol{x}), \sigma_t)$.

Now, let us denote by $r(\alpha, t)$ the performance of an optimization method α on a task t, for instance, a common example for r is the regret of the final incumbent solution. To compare two different methods α_A and α_B, the standard practice is to compare $r(\alpha_A, t_i)$ with $r(\alpha_B, t_i)$ on a set of hand-picked tasks $t_i \in \{t_0, \ldots t_M\}$. However, to draw statistically more significant conclusion we would ideally

like to integrated over all tasks:

$$S_{p(t)}(\alpha) = \int r(\alpha, t)p(t)dt. \tag{1}$$

Unfortunately, the above integral is intractable as $p(t)$ is unknown. The main contribution of this paper is to approximate $p(t)$ with a generative meta-model $\hat{p}(t \mid \mathcal{D})$ based on some off-line generated data $\mathcal{D} = \left\{\{(\boldsymbol{x}_{tn}, y_{tn})\}_{n=1}^{N}\right\}_{t=1}^{T}$. This enables us to sample an arbitrary amount of tasks $t_i \sim \hat{p}(t \mid \mathcal{D})$ in order to perform a Monte-Carlo approximation of Equation 1. As we show in the experimental section, using samples from the meta-model allows us to draw statistically significant insights about the performance of the HPO methods that could not be reached otherwise, where the limiting factor is only the computational overhead of the HPO method.

3.2 Meta-Model for Task Generation

In order to reason across tasks, we define a probabilistic encoder $p(\mathbf{h}_t \mid \mathcal{D})$ that learns a latent representation $\mathbf{h}_t \in \mathbb{R}^Q$ of a task t. More precisely, we use Bayesian GP-LVM (Titsias and Lawrence, 2010) which assumes that the target values that belong to the task t, stacked into a vector $\boldsymbol{y}_t = (y_{t1}, \ldots, y_{tN})$ follow the generative process:

$$\boldsymbol{y}_t = g(\mathbf{h}_t) + \epsilon, \quad g \sim \mathcal{GP}(0, k), \quad \epsilon \sim \mathcal{N}(0, \sigma^2), \tag{2}$$

where k is the covariance function of the GP. By assuming that the latent variable $\mathbf{h}_t$ has an uninformative prior $\mathbf{h}_t \sim \mathcal{N}(0, \mathbf{I})$, the latent embedding of each task is inferred as the posterior distribution $p(\mathbf{h}_t \mid \mathcal{D})$. The exact formulation of the posterior distribution is intractable, but following the variational inference presented in (Titsias and Lawrence, 2010), we can estimate a variational posterior distribution $q(\mathbf{h}_t) = \mathcal{N}(m_t, \Sigma_t) \approx p(\mathbf{h}_t \mid \mathcal{D})$ for each task t.

Now, similar to Multi-Task Bayesian Optimization (Swersky et al., 2013; Springenberg et al., 2016), we define a probabilistic model for the objective function $p(y_t \mid \boldsymbol{x}, \mathbf{h}_t)$ across tasks which gets as an additional input a task embedding based on our probabilistic encoder. Following (Springenberg et al., 2016), we use a Bayesian neural network with M weight vectors $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M\}$ to model

$$\begin{aligned} p(y_t \mid \boldsymbol{x}, \mathbf{h}_t, \mathcal{D}) &= \int p(y_t \mid \boldsymbol{x}, \mathbf{h}_t, \boldsymbol{\theta})p(\theta \mid \mathcal{D})d\boldsymbol{\theta} \\ &\approx \frac{1}{M}\sum_{i=1}^{M} p(y_t \mid \boldsymbol{x}, \mathbf{h}_t, \boldsymbol{\theta}_i). \end{aligned} \tag{3}$$

where $\theta_i \sim p(\theta \mid \mathcal{D})$ is sampled from the posterior of the neural network weights.

By approximating $p(y_t \mid \boldsymbol{x}, \mathbf{h}_t) = \mathcal{N}\big(\mu(\boldsymbol{x}, \mathbf{h}_t), \sigma^2(\boldsymbol{x}, \mathbf{h}_t)\big)$ to be Gaussian, we can compute the predictive mean and variance by (Springenberg et al., 2016):

$$\begin{aligned} \mu(\boldsymbol{x}, \mathbf{h}_t) &= \frac{1}{M}\sum_{i=1}^{M} \hat{\mu}(\boldsymbol{x}, \mathbf{h}_t \mid \boldsymbol{\theta}_i), \\ \sigma^2(\boldsymbol{x}, \mathbf{h}_t) &= \frac{1}{M}\sum_{i=1}^{M} \big(\hat{\mu}(\boldsymbol{x}, \mathbf{h}_t \mid \boldsymbol{\theta}_i) - \mu(\boldsymbol{x}, \mathbf{h}_t)\big)^2 + \hat{\sigma}^2_{\boldsymbol{\theta}_i}. \end{aligned}$$

where $\hat{\mu}(\boldsymbol{x}, \mathbf{h}_t \mid \boldsymbol{\theta}_i)$ and $\hat{\sigma}^2_{\boldsymbol{\theta}_i}$ are the output of a single neural network with parameters $\boldsymbol{\theta}_i$[1]. To get a set of weights $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M\}$, we use stochastic gradient Hamiltonian Monte-Carlo (Chen et al., 2014) to sample $\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta}, \mathcal{D})$ from:

$$p(\boldsymbol{\theta}, \mathcal{D}) = \frac{1}{N}\sum_{n}\frac{1}{H}\sum_{j}\log p(y_n \mid \boldsymbol{x}_n, \mathbf{h}_{nj})$$

with $N = |D|$ and H the number of samples we draw from the latent space $\mathbf{h}_{tj} \sim q(\mathbf{h}_t)$.

[1]Note that we model an homoscedastic noise, because of that, $\hat{\sigma}^2_{\boldsymbol{\theta}_i}$ does not depend on the input

3.3 Sampling New Tasks

In order to generate a new task $t_\star \sim \hat{p}(t \mid \mathcal{D})$, we need the associated objective function $f_{t_\star}$ in a parameteric from such that we can evaluate it later on any $\boldsymbol{x} \in \mathcal{X}$.

Given the meta-model above, we perform the following steps: (i) we sample a new latent task vector $\mathbf{h}_{t_\star} \sim q(\mathbf{h}_t)$; (ii) given $\mathbf{h}_{t_\star}$ we pick a random $\boldsymbol{\theta}_i$ from the set of weights $\{\boldsymbol{\theta}_1, \dots \boldsymbol{\theta}_M\}$ of our Bayesian neural network and set the new task to be $f_{t_\star}(\boldsymbol{x}) = \hat{\mu}(\boldsymbol{x}, \mathbf{h}_{t_\star} \mid \boldsymbol{\theta}_i)$.

Note that using $f_{t_\star}(\boldsymbol{x})$ makes our new task unrealisticly smooth. Instead, we can emulate the typical noise appearing in HPO benchmarks by returning $y_{t_\star}(\boldsymbol{x}) \sim \mathcal{N}\big(\hat{\mu}(\boldsymbol{x}, \mathbf{h}_{t_\star} \mid \boldsymbol{\theta}_i), \hat{\sigma}^2_{\boldsymbol{\theta}_i}\big)$, which can be done at an insignificant cost.

4 Profet

We now present PROFET, a benchmarking suite for HPO methods. The following section describes how we collected the data to train our meta-model for three typical HPO problems. After training, we drew $T = 1000$ different tasks for each problem from our meta-model and, as described above, provide a noisy and noiseless version of each task. Second, we discuss two ways that are commonly used in the literature to assess and aggregate the performance of HPO methods across tasks. To reproduce our experiments as well as benchmarking and developing new HPO methods, an open-source implementation of PROFET is available here: XXX

4.1 Data Collection

We consider three different HPO problems, two for classification and one for regression, with varying dimensions D. For classification we considered a support vector machine (SVM) with $D = 2$ hyperparameters and a feed forward neural network (FC-Net) with $D = 6$ hyperparameters on 16 OpenML (Vanschoren et al., 2014) tasks. We used gradient boosting (XGBoost)[2] with $D = 8$ hyperparameters for regression on 11 different UCI datasets (Lichman, 2013).

For further details about the datasets and the configuration spaces see Appendix A. To make sure that our meta-model learns a descriptive representation we need a solid coverage over the whole input space. For that we drew $100D$ pseudo randomly generated configurations from a Sobol grid (Sobol, 1967).

The neural network architecture for our meta-model consisted of 3 fully connected layers with 500 units each and tanh activation functions. We used Bayesian GP-LVM[3](Titsias and Lawrence, 2010) with a Matern52 kernel to learn a $Q = 5$ dimensional latent space for the task description. We show some qualitative example of our probabilistic encoder in Section 5.1.

For a more realistic scenario, we apply the same machinery to model the cost in terms of computation time for evaluating a hyperparameter configuration. This allows us to use time rather than function evaluations as budget. Besides that, it also enables future work to benchmark or develop HPO methods that explicitly take the cost into account (e. g. EIperSec by Snoek et al. (2012)).

4.2 Performance Assessment

To assess the performance of a HPO method, we consider two different ways that are commonly used in the literature to aggregate over tasks. First, we measure the **runtime** $r(\alpha, t, y_{target})$ a HPO method α needs to find a configuration that achieves a performance that is equal or lower than a certain target value y_{target} on task t (Hansen et al., 2016a). Here we define runtime either in terms of function evaluations or estimated wall-clock time predicted by our meta-model. Using a fixed target approach allows us to draw quantitative statements, such as: *method A is, on average, twice as fast than method B*. See Hansen et al. (2016a) for a more detailed discussion.

The runtime of a HPO method strongly depends on the target value. If we set these target values too small, the benchmarks become too challenging and only few methods will converge. On the other hand, larger target values lead to easy benchmarks where every method converges after few iterations.

[2]We used the implementation from Chen and Guestrin (2016)

[3]We used the implementation from GPy (2012)

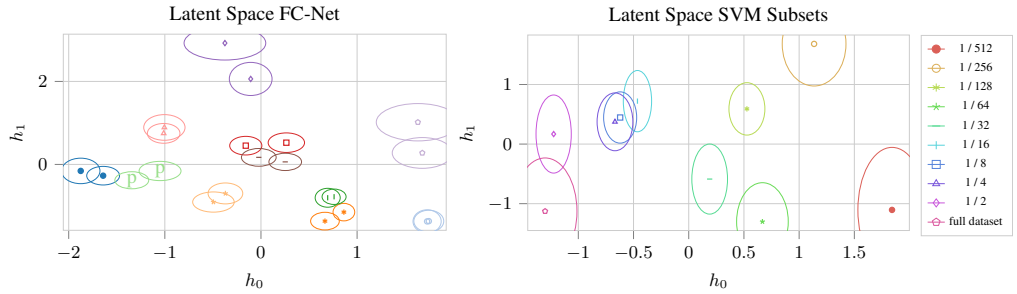

Figure 3: Latent space representations of two datasets of tasks. *Left*: Representation of eleven pairs of datasets generated by partitioning eleven datasets from the fully connected networks benchmark detailed in Section 4.1. Pairs of tasks are represented with the same colour. The mean of the task are represented with dots and stars. The ellipses represent 4 standard deviations around the mean of the tasks. *Right*: Latent space learned for a model where the input tasks are generated by training a support vector machine on subsets of a target dataset. The objective function is approximated by a random forest surrogate from Klein et al. (2017a).

Because of that, we average across target values with a different complexity. For that we evaluate the Sobol grid from above on each generated tasks and use the corresponding function values as targets, which, with the same argument as described in Section 4.1, provides a good coverage of the error surface. To aggregate the runtime we use the empirical cumulative distribution function (ECDF) (Moré and Wild, 2009), which intuitively, shows for each budget on the x-axis the fraction of solved task and target pairs on the y-axis.

Another common way to compare different HPO methods is to compute the **average ranking score** in every iteration and for every task (Bardenet et al., 2013). We follow the procedure described by Feurer et al. (2015b) and compute the average ranking score as follows: assuming we run K different HPO methods M times for each task, we draw a bootstrap sample of 1000 runs out of the K^M possible combinations. For each of these samples, we compute the average fractional ranking (ties are broken by the average of the ordinal ranks) after each iteration. At the end all the assigned ranks are further averaged over all tasks. Note that averaged ranks are a relative performance measurement and can decrease for one method if another method's performance increases.

5 Experiments

In this section we present: (i) some qualitative insights of our meta-model by showing how it is able to coherently represent a sets of tasks in its latent space, (ii) and illustration of why PROFET helps to obtain statistically meaningful results in HPO benchmarks and (iii) a comparison of various methods from the literature on our new benchmark suite. In particular, we show results for the following state-of-the-art Bayesian optimization (BO) methods for HPO as well as two popular evolutionary algorithms for general continuous black-box optimization problems:

- BO with Gaussian processes (BO-GP) (Jones et al., 1998). We used expected improvement as acquisition function and marginalize over the Gaussian process' hyperparameters as described by Snoek et al. (2012).
- SMAC (Hutter et al., 2011): which is a variant of BO that uses random forests to model the objective function and stochastic local search to optimize expected improvement.
  We use the implementation from https://github.com/automl/SMAC3.
- The BO method TPE by Bergstra et al. (2011) which models the density of good and bad configurations in the input space with a kernel density estimators. We used the implementation provided from the Hyperopt package (Komer et al., 2014)
- BO with Bayesian neural networks (BOHAMIANN) as described by Springenberg et al. (2016). To avoid introducing any bias, we used a different architecture with less parameters (3 layers, 50 units in each) than we used for our meta-model (see Section 3).
- Differential Evolution (DE) (Storn and Price, 1997) (we used our own implementation) with rand1 strategy for the mutation operators and a population size of 10.

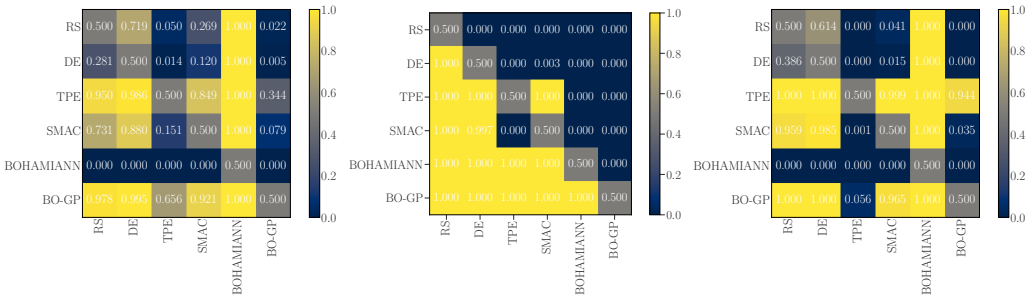

Figure 4: Heatmaps of the p-values of the pairwise comparisons across the methods in the three scenarios using a Mann-Whitney U test. Small p-values should be interpreted as the test finding evidence that the method in the column improves the method in the row. Using tasks from our meta-model instead lead to results that are very close to using a sufficient large set of tasks from the original distribution. *Left*: results with 1000 real tasks. *Middle*: results with 9 reals tasks. *Right:* results with 1000 tasks generated from our meta-model.

- Covariance Matrix Adaption Evolution Strategy (CMA-ES) by Hansen (2006) where we used the implementation from https://github.com/CMA-ES/pycma
- Random Search (RS) (Bergstra and Bengio, 2012) which samples configurations uniformly at random.

For BO-GP, BOHAMIANN and RS we used the implementation provided by the RoBO package (Klein et al., 2017b). For every method we used the provided default hyperparameter settings.

5.1 Tasks Representation in the Latent Space

We demonstrate the interpretability of the learned latent representations of tasks in two examples. For the first experiment we used the fully connected network benchmark described in Section 4.1. To visualize that our meta-model learns a meaningful latent space, we doubled 11 out of the 18 original tasks to train the model by splitting each one of them randomly in two of the same size. Thereby, we guarantee that there are pairs of tasks that are similar to each other. In Figure 3 (left), each color represents the partition of the original task and each ellipse represents the mean and four times the standard deviation of the latent task representations. One can see that the closest neighbour of each task is the other task that belongs to the same original task.

The second experiment targets multi-fidelity experiments that arise when a machine learning model needs to be trained on a very large dataset and approximate versions of the target objective are generated by considering subsamples of different sizes. For this experiment we used the SVM surrogate for different datasets subsets from Klein et al. (2017a). The surrogate consists of a random forest trained on a grid of hyperparameter configurations of a SVM evaluated on different subsets of the training data. In particular, we defined the following subsets: $\{1/512, 1/256, 1/128, 1/64, 1/32, 1/16, 1/8, 1/4, 1/2, 1\}$ as tasks and sampled 100 configurations per task to train our meta-model. Figure 3 (right) shows the latent space of the trained meta-model. As one can see, the latent representation of the model is able to capture a latent representation between tasks where similar data subsets are also close in the latent space. In particular, the first latent dimension coherently captures the sample size, which is learned using exclusively the correlation between the datasets and with no further information about their size.

5.2 Benchmarking with PROFET

Comparing HPO methods using a small number of instances affects our ability to find significant results when differences exist. To illustrate this we consider a family of objective functions that are variations of the function $f(x) = ((\alpha x - 2)^2)\sin(\beta x - 4)$ for parameters α and β. We generated 1000 tasks by uniformly sampling random α and β in $[0, 1]$ and compared six HPO methods: RS, DE, TPE, SMAC, BOHAMIANN and BO-GP (we left CMA-ES out because the python version does not support 1-dimensional optimization problems).

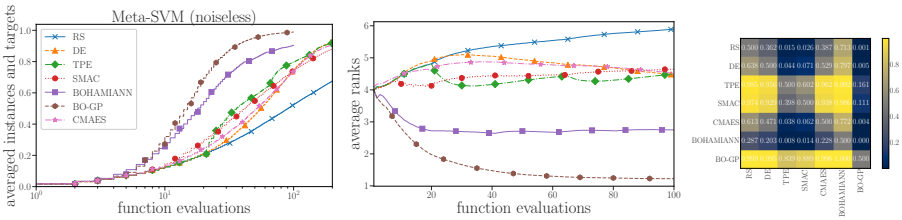

Figure 5: Comparison of various different methods on 1000 tasks sampled from the Meta-SVM benchmark without noise. See Section D in the supplemental material for the results on all benchmark problems. *Left:* the ECDF for the runtime. *Middle*: the ranking of each method averaged across all tasks. *Right:* the p-values of the pairwise Mann-Whitney-U test for the averaged performances after 100 function evaluation. Interpretation as detailed in Section 5.3.

Figure 4 (top) shows the p-values of all pairwise comparisons with the null hypothesis 'Method$_{column}$ achieves a higher error after 50 function evaluations averaged over 20 runs than Method$_{row}$' for the Mann-Whitney U test. Squares in the figure with a p-value smaller than 0.05 are comparisons in which with a 95% confidence we have evidence to show that the method in the column is better that the method in the row (we have evidence to reject the null hypothesis). The comparison shows that BO-GP is the superior method although significant results are not observed when compared with TPE. We repeat the same experiment but with 9 randomly selected tasks (Figure 4, center) where for each tasks we performed 2220 runs of every method, and then computed the average of groups of 20 runs, such that we obtain 1000 samples per method to compute the statistical test. One can see, while the results are statistically significant, they are misleading, for example, BOHAMIANN is dominating all other methods (except BO-GP), whereas it is significant worse than all other methods if consider all 1000 tasks.

To solve this issue we use PROFET. We first train the meta-model on the 9 selected tasks and then use it to produce 1000 new tasks. See Appendix C for an visualization. Next, we use the 1000 generated tasks to run the comparison of the HPO methods. Results are shown in Figure 4 (bottom). The heatmap of statistical comparisons reaches very similar conclusions to those obtained with the original 100 tasks, contrary to what happen when we do the comparisons with 9 tasks only (i. e. p-values are closer to the original ones). Using samples from the meta-model allows us to discover statistical differences across HPO methods that are more consistent with the real differences than what we observe if we only use a small number of tasks.

5.3 Comparing State-of-the-art HPO Methods

We conducted 20 independent runs for each method on each task with a different random seed. Each method had a budget of 200 function evaluations for each task, except for BO-GP and BOHAMIANN, where, due to the computational overhead of selecting a new configuration, we were only able to perform 100 function evaluations.

For the noiseless Meta-SVM benchmark, in Figure 5 we show the full ECDF curves, the average ranking and the heatmap of statistical comparisons. The results for all other datasets are shown in Section E of the supplementary material. We can make the following observations:

- Given enough budget, all methods are able to outperform RS. BO approaches can exploit their internal model of the objective function faster such that they start to outperform RS earlier than evolutionary algorithms (DE, CMA-ES). Thereby, more sophisticated models, such as Gaussian processes or Bayesian neural network are more *data efficient* than somewhat simpler methods such as random forests or kernel density estimators.
- The performance of BO methods that model the objective function (BO-GP, BOHAMIANN, SMAC) instead of just the distribution of the input space (TPE) decays if we evaluate the function through noise. Also evolutionary algorithms struggle with noise.
- Standard BO (BO-GP) works superior on these benchmarks but its performance decays rapidly with the number of dimensions.

- Runner-up is BOHAMIANN which works slightly worse than BO-GP but seems to suffer less under noisy function values. Note that this result can only be achieved by using PROFET as we could not have reached this conclusions with the original datasets.
- Given a sufficient budget, DE starts to outperform CMA-ES as well as BO with simpler (and cheaper) models of the objective function (SMAC, TPE), making it a competitive baseline particularly for higher dimensional benchmarks.

6 Discussion

While PROFET enables thorough benchmarking of HPO algorithms and we show empirical evidence that captures the properties of the original benchmarks, due to the underlying statistical model, there is no guarantee that results on PROFET translate one-to-one to the original benchmarks. Nevertheless, we believe that PROFET, and surrogate benchmarks in general, provide much more realistic use-cases than synthetic functions, such as for example Branin, which are usually used during the developing of new HPO methods. Overall, we hope that future work uses PROFET to rapidly perform reliable experiments during developing and only perform the final evaluation of a new developed method on the expensive original benchmarks.

7 Conclusions and future work

We presented PROFET, a new tool for benchmarking HPO algorithms in machine learning. The key idea is to use a generative meta-model that is trained using a set of available tasks and then used to produce new tasks, possibly perturbed by noise. The new synthetically generated tasks retain the properties of the original tasks but can be evaluated inexpensively, which represents a major advance to speed up comparisons of HPO methods. The model proposed in this work is a combination of a GP-LVM and a BNN. In a battery of experiments we have illustrated the representation power of the meta-model used in PROFET and its utility when comparing HPO methods in families of problems in which only a few tasks are available. Although in this work we have focused on the comparison of HPO methods, the same idea can be generalized to other optimization problems.

In future work we will consider multi-fidelity benchmarks (Klein et al., 2017a; Kandasamy et al., 2017; Klein et al., 2017c) where cheap but approximate fidelities of the objective function are available, such as learning curves or dataset subsets. Other generative models are also worth to be investigated. Furthermore, since PROFET also provides gradient information of the parametric tasks it might be interesting as a training distribution for learning-to-learn approaches (Chen et al., 2017).

References

Bardenet, R., Brendel, M., Kégl, B., and Sebag, M. (2013). Collaborative hyperparameter tuning. In *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Advances in Neural Information Processing Systems (NIPS'11)*.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*.

Chen, T., Fox, E., and Guestrin, C. (2014). Stochastic gradient Hamiltonian Monte Carlo. In *Proceedings of the 31th International Conference on Machine Learning, (ICML'14)*.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.

Chen, Y., Hoffman, M. W., Colmenarejo, S. G., Denil, M., Lillicrap, T. P., Botvinick, M., and de Freitas, N. (2017). Learning to learn without gradient descent by gradient descent. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*.

Chen, Y., Huang, A., Wang, Z., Antonoglou, I., Schrittwieser, J., Silver, D., and de Freitas, N. (2018). Bayesian optimization in AlphaGo. *arXiv:1812.06855 [cs.LG]*.

Dai, Z., Álvarez, M. A., and Lawrence, N. (2017). Efficient modeling of latent information in supervised learning using gaussian processes. In *Proceedings of the 30th International Conference on Advances in Neural Information Processing Systems (NIPS'17)*.

Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., and Leyton-Brown, K. (2013). Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS Workshop on Bayesian Optimization (BayesOpt'13)*.

Eggensperger, K., Hutter, F., Hoos, H., and Leyton-Brown, K. (2015). Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the 29th National Conference on Artificial Intelligence (AAAI'15)*.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015a). Efficient and robust automated machine learning. In *Proceedings of the 28th International Conference on Advances in Neural Information Processing Systems (NIPS'15)*.

Feurer, M., Springenberg, T., and Hutter, F. (2015b). Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the 29th National Conference on Artificial Intelligence (AAAI'15)*.

GPy (since 2012). GPy: A gaussian process framework in python. `http://github.com/SheffieldML/GPy`.

Hansen, N. (2006). The CMA evolution strategy: a comparing review. In *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*. Springer Berlin Heidelberg.

Hansen, N., Auger, A., Brockhoff, D., Tusar, D., and Tusar, T. (2016a). COCO: performance assessment. *arXiv:1605.03560 [cs.NE]*.

Hansen, N., Auger, A., Mersmann, O., Tušar, T., and Brockhoff, D. (2016b). COCO: A platform for comparing continuous optimizers in a black-box setting. *arXiv:1603.08785 [cs.AI]*.

Hutter, F., Hoos, H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11)*.

Hutter, F., Kotthoff, L., and Vanschoren, J., editors (2018). *Automatic Machine Learning: Methods, Systems, Challenges*. Springer.

Jones, D., Schonlau, M., and Welch, W. (1998). Efficient global optimization of expensive black box functions. *Journal of Global Optimization*.

Kandasamy, K., Dasarathy, G., Schneider, J., and Póczos, B. (2017). Multi-fidelity bayesian optimisation with continuous approximations. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*.

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. (2017a). Fast Bayesian hyperparameter optimization on large datasets. In *Electronic Journal of Statistics*.

Klein, A., Falkner, S., Mansur, N., and Hutter, F. (2017b). Robo: A flexible and robust bayesian optimization framework in python. In *NIPS Workshop on Bayesian Optimization (BayesOpt'17)*.

Klein, A., Falkner, S., Springenberg, J. T., and Hutter, F. (2017c). Learning curve prediction with Bayesian neural networks. In *International Conference on Learning Representations (ICLR'17)*.

Komer, B., Bergstra, J., and Eliasmith, C. (2014). Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn. In *ICML 2014 AutoML Workshop*.

Lichman, M. (2013). UCI machine learning repository.

Moré, J. J. and Wild, S. M. (2009). Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*.

Perrone, V., Jenatton, R., Seeger, M., and Archambeau, C. (2018). Scalable hyperparameter transfer learning. In *Proceedings of the 31th International Conference on Advances in Neural Information Processing Systems (NIPS'18)*.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. (2017). Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*.

Shahriari, B., Swersky, K., Wang, Z., Adams, R., and de Freitas, N. (2016). Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NIPS'12)*.

Sobol, I. M. (1967). Distribution of points in a cube and approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*.

Springenberg, J. T., Klein, A., Falkner, S., and Hutter, F. (2016). Bayesian optimization with robust bayesian neural networks. In *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NIPS'16)*.

Storn, R. and Price, K. (1997). Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*.

Swersky, K., Snoek, J., and Adams, R. (2013). Multi-task Bayesian optimization. In *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NIPS'13)*.

Titsias, M. and Lawrence, N. (2010). Bayesian Gaussian process latent variable model. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS'10)*.

van Rijn, J. and Hutter, F. (2018). Hyperparameter importance across datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'18)*.

Vanschoren, J., van Rijn, J., Bischl, B., and Torgo, L. (2014). OpenML: Networked science in machine learning. *SIGKDD Explorations*.

Ying, C., Klein, A., Real, E., Christiansen, E., Murphy, K., and Hutter, F. (2019). NAS-Bench-101: Towards reproducible neural architecture search. *arXiv:1902.09635 [cs.LG]*.

Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR'17)*.